Docket No.: L&L-I0051

I hereby certify that this correspondence is being deposited with the United States Postal Service as First Class Mail in an envelope addressed to the Assistant Commissioner for Patents, Washington, D.C. 20231, on the date indicated below.

By: _____ Date: September 11, 2001

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant	:	Wolfgang Ecker et al.
Applic. No.	:	09/935,355
Filed	:	August 22, 2001
Title	:	Method, Computer Program Product, Programmed Data Medium, and Computer System for Revising a Computer Program Written in a Programming Language

CLAIM FOR PRIORITY

Hon. Commissioner of Patents and Trademarks,
Washington, D.C. 20231

Sir:

Claim is hereby made for a right of priority under Title 35, U.S. Code, Section 199, based upon the German Patent Application 100 41 111.8, filed August 22, 2000.

A certified copy of the above-mentioned foreign patent application is being submitted herewith.

Respectfully submitted,

For Applicants WERNER H. STEMER
REG. NO. 34,956

Date: September 11, 2001

Lerner and Greenberg, P.A.
Post Office Box 2480
Hollywood, FL 33022-2480
Tel: (954) 925-1100
Fax: (954) 925-1101

/kf

BUNDESREPUBLIK DEUTSCHLAND

Prioritätsbescheinigung über die Einreichung einer Patentanmeldung

Aktenzeichen: 100 41 111.8

Anmeldetag: 22. August 2000

Anmelder/Inhaber: Infineon Technologies AG, München/DE

Bezeichnung: Verfahren zum Überarbeiten eines in einer Programmiersprache verfaßten Computerprogramms

IPC: G 06 F 9/45

Die angehefteten Stücke sind eine richtige und genaue Wiedergabe der ursprünglichen Unterlagen dieser Patentanmeldung.

München, den 4. September 2001
Deutsches Patent- und Markenamt
Der Präsident
Im Auftrag

Wehner

A 9161
02/00
EDV-L

Beschreibung

Verfahren zum Überarbeiten eines in einer Programmiersprache verfaßten Computerprogramms

Fehler im Quell- oder Sourcecode von Computerprogrammen führen zu Funktionsstörungen des Computerprogramms bzw. des Computers, auf dem das Computerprogramm ausgeführt wird. Wegen der oft hohen Komplexität von Computerprogrammen sind Fehler im Quellcode häufig nur mit Schwierigkeiten zu lokalisieren.

Für große Computerprogramme ist es weiterhin wichtig, daß sie gut lesbar sind, so daß auch Außenstehende den Quellcode - ggf. nach vielen Jahren - lesen und nachvollziehen können. Die Lesbarkeit von Quellcode kann z.B. dadurch erhöht werden, daß letzterer streng nach gewissen Regeln oder Kodierstilen gestaltet ist, die die Lesbarkeit erhöhen. Sollte ein Computerprogramm eine Verletzung einer Kodierregel aufweisen, dann sollte diese nach Möglichkeit beseitigt werden.

Dabei ist es zweckmäßig, die manuelle Fehlersuche mit Hilfe geeigneter Computerprogramme zu unterstützen. Bekannt sind sogenannte Beautifier, die Einrückungen und Zeilenumbrüche in Quellcode einfügen bzw. entfernen. Einrückungen sind jedoch lediglich Trennzeichen, durch deren Veränderung keine Fehler behoben werden. Beautifier beachten lediglich reine Formatierungsregeln.

Ferner gibt es Computerprogramme zur Überprüfung der Verletzung von Kodierstilen in Computerprogrammen, aber auch in Hardware-Modellen, die in einer Hardware-Beschreibungssprache wie VHDL verfaßt sind. Diese Programme sind (lediglich) in der Lage, auf ermittelte Regelverletzungen hinzuweisen.

In speziellen Fällen kann es vorkommen, daß Verletzungen von Kodierregeln nicht vermieden werden können, etwa wenn externer Code eingebunden wird, der nicht den internen Kodierre-

geln folgt. In solchen Fällen ist es für den Programmierer lästig, wenn diese unumgänglichen Verletzungen, die im Quellcode verbleiben sollen, bei jeder erneuten Überprüfung als Verletzungen gemeldet werden, woraufhin sie jedesmal als zu ignorieren gekennzeichnet werden müssen bzw. eine vorgeschlagene Änderung verworfen werden muß.

Aufgabe der Erfindung ist es, die bekannten Computerprogramme zur Überprüfung von Computerprogrammen zu verbessern.

Nach einem ersten Aspekt der Erfindung wird die Aufgabe durch ein Verfahren nach Anspruch 1, ein Computerprogrammprodukt nach Anspruch 6 oder 7, einen Datenträger nach Anspruch 8 sowie durch ein Computersystem nach Anspruch 9 oder 10 gelöst. Unter Computerprogrammprodukt wird dabei das Computerprogramm als handelbares Produkt verstanden, in welcher Form auch immer, z.B. auf einem computerlesbaren Datenträger, über ein Netz verteilt, etc.

Erfindungsgemäß wird ein Computerprogramm eingesetzt, das zunächst, wie bekannt, Regelverletzungen erkennt. Eine Regelverletzung kann dabei ein echter Programmierfehler oder die Verletzung eines Kodierstils oder einer sonstigen Konsistenz-, Syntax- oder Grammatik-Regel oder einer lexikalischen Regel sein. Die Regelverletzungen werden jedoch nicht nur ausgegeben, sondern es werden auch eine oder mehrere mögliche Korrekturen des Computerprogramms berechnet. Aus den möglichen Korrekturen wird eine Korrektur automatisch oder interaktiv ausgewählt und das Computerprogramm wird gemäß der ermittelten Korrektur sofort oder nach nochmaliger interaktiver Zustimmung geändert.

Die Erfindung schafft eine automatische grammatikalische, syntaktische und semantische Korrektur von Computerprogrammen. Ein Vorteil der Erfindung besteht daher in dem erheblichen Zeitgewinn gegenüber einer manuellen Korrektur. Ferner werden alle Änderungen durch den automatisierten Charakter

konsistent durchgeführt. Dadurch wird ferner der Aufwand für die Überprüfung von Computerprogrammen erheblich reduziert. Ein Computerprogramm kann somit in jeder einzelnen Version überprüft werden.

Die Erfindung kann für beliebige Programmiersprachen eingesetzt werden. Auch kann sie für Hardware-Modelle eingesetzt werden, die in Hardware-Beschreibungssprachen verfaßt sind.

In einer vorteilhaften Weiterbildung der Erfindung gemäß ihrem ersten Aspekt kann ein entstehendes Computerprogramm bereits während einer sukzessiven Eingabe auf Verletzungen von vorgegebenen Regeln durchsucht werden. Sobald ein Ausdruck derart abgeschlossen ist, daß eine Verletzung erkannt werden kann, kann diese noch während der Eingabe graphisch kenntlich gemacht werden. Ein Programmierer kann dadurch sofort bei der Eingabe Verletzungen von Regeln erkennen.

Einfache Regelverletzungen mit eindeutigen Korrekturen können - sobald sie eindeutig zu erkennen sind - unmittelbar während der Eingabe korrigiert werden.

Gemäß einem zweiten Aspekt der Erfindung wird die Aufgabe durch ein Verfahren nach Anspruch 15, ein Computerprogrammprodukt nach Anspruch 23 oder 24, einen Datenträger nach Anspruch 25 sowie durch ein Computersystem nach Anspruch 26 oder 27 gelöst.

Es wird ein Computerprogramm eingesetzt, das zunächst, wie bekannt, Regelverletzungen erkennt und/oder gemäß dem ersten Aspekt der Erfindung Korrekturen ermittelt und ggf. selbstätig ausführt. Eine Regelverletzung kann dabei ein echter Programmierfehler oder die Verletzung einer Kodierregel oder einer sonstigen Konsistenz-, Syntax- oder Grammatik-Regel oder einer lexikalischen Regel sein. Nach dem zweiten Aspekt der Erfindung können dabei jedoch Verletzungen gesondert oder in-

teraktiv definiert werden, die bei der Analyse automatisch ignoriert werden.

Die Überprüfung unvermeidlicher Verletzungen wird somit vermieden. Dadurch wird der Aufwand für die Überprüfung von Computerprogrammen reduziert.

Die Erfindung kann für beliebige Programmiersprachen eingesetzt werden. Auch kann sie für Hardware-Modelle eingesetzt werden, die in Hardware-Beschreibungssprachen verfaßt sind.

Besonders günstig ist es, die zu ignorierenden Verletzungen durch ihre Stellung in der Programm-Hierarchie zu definieren, oder durch Angabe ihrer Deklarationsumgebung oder sonstiger Gebiete im Code. Eine derartige Spezifizierung ist robust gegen gewisse Änderungen des analysierten Computerprogramms, etwa das Verschieben von Zeilen durch Einfügen oder Auslassen, oder das Entfernen gewisser Blöcke des Programms. Gleiches gilt, wenn Verletzungen von Kodierregeln für eine Klasse von Konstrukten generell zugelassen werden.

Weitere vorteilhafte Weiterbildungen der Erfindung sind in den Unteransprüchen angegeben.

Im Folgenden wird die Erfindung anhand von Ausführungsbeipielen näher erläutert, die in den Figuren schematisch dargestellt sind. Im Einzelnen zeigt:

Fig. 1 eine schematische Darstellung des erfindungsgemäßen Computersystems;

Fig. 2 eine schematische Darstellung eines erfindungsgemäßen Verfahrens nach dem ersten Aspekt der Erfindung; und

Fig. 3 eine schematische Darstellung eines erfindungsgemäßen Verfahrens nach dem zweiten Aspekt der Erfindung.

Fig.1 zeigt einen Computer 10 mit einer Tastatur 12, die als Eingabemittel dienen kann. Ferner weist der Computer 10 ein Diskettenlaufwerk 14 auf, das sowohl zur Ein- als auch zur Ausgabe dienen kann. Im Computer 10 befindet sich ein Speicher 16. Dieser kann beispielsweise eine Festplatte oder auch ein Arbeitsspeicher sein. Der Computer 10 weist eine zentrale Rechen- oder Verarbeitungseinheit 18 auf (CPU, central processing unit). Weiterhin dienen der Ausgabe ein Monitor 20 und ein Drucker 22. Der Computer 10 kann zur Ein- und Ausgabe auch an ein Netzwerk (nicht gezeigt) angeschlossen sein.

Fig. 2 zeigt schematisch die Schritte des auf dem Computer 10 ausgeführten Verfahrens. Zunächst wird das zu überprüfende Computerprogramm etwa von einer Diskette im Diskettenlaufwerk 14 in den RAM 16 geladen.

Das Computerprogramm ist in einer Programmiersprache geschrieben, beispielsweise in VHDL (siehe unten). Zu der Grammatik von VHDL möge es noch weitere Kodierkonventionen geben, die zwar nicht zwingend durch VHDL vorgegeben sind, aber die Lesbarkeit oder Konsistenz des Computerprogramms erhöhen. Diese Regeln beschränken die Grammatik von VHDL.

Die CPU 18 liest das zu untersuchende Computerprogramm aus dem Arbeitsspeicher 16 und durchsucht es auf Verletzungen von vorgegebenen Regeln der Programmiersprache und der weiteren Kodierkonventionen. Für jede gefundene Verletzung einer vorgegebenen Regel wird mindestens eine mögliche Korrektur im Computerprogramm berechnet. Hierfür wird die gefundene Verletzung formal analysiert und auf der Basis des Analyseergebnisses werden Korrekturmöglichkeiten bestimmt. Zu diesem Zweck sind jedem Analyseergebnis eine oder mehrere entsprechende Korrekturmöglichkeiten zugeordnet. Die Korrekturmöglichkeiten werden auf dem Monitor 20 oder über das Netzwerk zur interaktiven Auswahl oder Bestätigung durch einen Benutzer ausgeben. Der Benutzer kann aus den verschiedenen Korrek-

turmöglichkeiten mit Hilfe der Tastatur 12 oder einer (nicht gezeigten) Maus oder mittels Sprachsteuerung auswählen.

Nach Wahl einer Korrekturmöglichkeit wird das zu überarbeitende Computerprogramm durch die Verarbeitungseinheit entsprechend geändert.

Dies wird mit allen ermittelten Regelverletzungen wiederholt. Nach Abschluß der Korrekturen wird das Computerprogramm von der CPU 18 wieder in den Arbeitsspeicher 16 geschrieben, von wo aus es den diversen Ausgabemitteln zugeleitet werden kann.

Im Folgenden werden anhand der Programmiersprache VHDL einige Beispiele für den Einsatz des anhand Fig. 2 erläuterten Verfahrens aufgezeigt. VHDL steht für "very high speed integrated circuits hardware description language". Es ist eine objektbasierte Programmiersprache, die speziell zum Beschreiben und Testen von Hardwarebausteinen wie ASICs entwickelt wurde.

1. Erkennung und Entfernung unbenutzter Objekte

Das Verfahren erkennt eine unbenutzte Variable v in dem unten dargestellten kurzen Programm und entfernt diese automatisch oder interaktiv.

Der Quellcode lautet zunächst:

```
function f( p : integer) return integer is
   variable v : integer;
begin
   return p+1 mod 100;
end f;
```

Hier wurde die Variable v zwar definiert, sie wird jedoch im weiteren Verlauf nicht verwendet. Nach der automatischen Korrektur erhält man:

```
function f( p : integer) return integer is
begin
   return p+1 mod 100;
end f;
```

Die überflüssige Variable v wurde eliminiert. Die Konsistenz des Programms ist wieder hergestellt.

2. Namenskonvertierung

Zusätzlich zu der Grammatik von VHDL möge es als Kodierkonvention noch die folgende lexikalische Namensregel geben, wonach Funktionen stets durch "_f" beendet werden und Konstanten durch "_c". Das Verfahren erkennt nun Namen, die nicht dem vorgegebenen Regelsatz entsprechen, und korrigiert diese.

Der Quellcode lautet zunächst:

```
function inc( number : integer) return integer is
begin
   return number + 1;
end inc;
```

Nach der automatischen Korrektur erhält man:

```
function inc_f( number_c : integer) return integer is
begin
   return number_c + 1;
end inc_f;
```

Die Konstanten und Funktionen sind nun sofort als solche zu erkennen, ohne daß es einer weitergehenden Analyse bedarf.

3. Vollständigkeit und Minimalität der Sensitivity-List

Das Verfahren überprüft in diesem Beispiel die Sensitivity-Liste (aus der Hardware-Beschreibungssprache VHDL) auf Vollständigkeit und Minimalität und korrigiert sie entsprechend. Die Sensitivity-Liste (im unten stehenden Beispiel "(a, b)" bzw. "(a, c)") ist eine Liste von Signalen, deren Änderung eine vorgegebene Aktion auslöst. Im unten stehenden korrigierten Beispiel löst eine Änderung von a oder c die Neuberechnung von d aus. "Vollständig" ist eine Sensitivity-Liste, wenn sie alle Signale enthält, deren Änderung die Aktion auslöst. "Minimal" ist sie, wenn sie keine überflüssigen Signale enthält.

Der Quellcode in diesem Beispiel lautet zunächst:

```
process (a, b) is
begin
    d <= a or c;
end process;
```

Hier ist die Sensitivity-Liste weder vollständig (es fehlt c) noch minimal (b ist überflüssig). Die Änderung von a oder c bewirkt eine Neuberechnung von d.

Nach der automatischen Korrektur erhält man:

```
process (a, c) is
begin
    d <= a or c;
end process;
```

Diese Zeilen sind konsistent, d.h. die Sensitivity-Liste ist minimal und vollständig.

4. Erweiterung von CASE-Anweisungen

Die CASE-Anweisung ist eine Verzweigungsanweisung in Abhängigkeit von einer endlichen Anzahl von möglichen Zuständen von Objekten, hier der Variablen "state". Diese kann im unten gezeigten Beispiel die Werte "red, green" oder "blue" annehmen. Jeder dieser Werte löst eine unterschiedliche Aktion aus. Das Verfahren erkennt solche sogenannten Finite State Machines (FSM) und fügt hier einen Default-Zweig in die entsprechende CASE-Anweisung ein, um Probleme beim Compilieren und Abbilden in Hardware zu vermeiden. Ein Default-Zweig gibt an, welche Aktion eintritt, falls die Variable "state" keinen der bereits definierten Zustände einnimmt. Es handelt sich hierbei um eine zusätzlich Grammatik-Regel, die die Konsistenz des Programms gewährleisten soll.

Der Quellcode lautet zunächst:

```
case state is
   when red    => a := 1;
   when green  => a := b;
   when blue   => b := a;
end case;
```

Nach der automatischen Korrektur erhält man:

```
case state is
   when red    => a := 1;
   when green  => a := b;
   when blue   => b := a;
   when others => null;
end case;
```

Die Konsistenz ist auch hier gewährleistet.

5. Interne Referenz zur aktuellen Bibliothek *work*

Das Verfahren erkennt Referenzen, die sich auf die eigene Bibliothek beziehen, aber nicht mit *work* benannt sind, wie sie es sein sollten. Da eine solche ungenaue Namensgebung Probleme für die Übersetzung bringen kann, benennt das Verfahren diese Referenzen automatisch um. Es handelt sich hier somit um eine weitere lexikalische Regel.

Der Quellcode lautet zunächst:

```
library atm;
use atm.atm_pack.all;

entity atm_top is
...
end atm_top;
```

Nach der automatischen Korrektur erhält man:

```
use work.atm_pack.all;

entity atm_top is
...
end atm_top;
```

Die Bibliothek hat jetzt den vorgegebenen Namen.

6. Separierung von Deklarationslisten

Das Verfahren löst kombinierte Deklarationen in übersichtlichere Einzeldeklarationen auf, beeinflußt somit die Syntax des Programms mit einer entsprechenden Syntax-Regel.

Der Quellcode lautet zunächst:

```
variable sum, arg, op: integer;
```

Nach der automatischen Korrektur erhält man:

```
variable sum: integer;
variable arg: integer;
variable op: integer;
```

Die syntaktische Umstellung der Deklaration hat hier den Effekt einer erhöhten Übersichtlichkeit.

7. Umwandlung von *positional association* in *named association*

Das Verfahren wandelt Anweisungen der Hardware-Beschreibungssprache VHDL, die die sogenannte *positional association* benutzen, in Anweisungen um, die die übersichtlichere *named association* benutzen. Bei der *positional association* ergibt sich die Zuordnung von aktuellen Parametern (hier z.B. "clk" oder "req") zu formalen Parametern aus der Stellung der Objekte im Ausdruck (hier an erster und dritter Position). Bei der *named association* wird explizit mit Namen angegeben, welches Objekt auf welchen Parameter abgebildet wird. Dadurch erhöht sich die Übersichtlichkeit gerade bei einer Vielzahl an Objekten erheblich.

Der Quellcode lautet zunächst:

```
h1: handshake
  port map (clk, res, req, ack, data);
```

Nach der automatischen Korrektur erhält man:

```
h1: handshake
  port map (
  clock       <= clk,
  reset       <= res,
  req_int     <= req,
  ack_out     <= ack,
```

```
data_in      <= data);
```

Hier liegt somit ein weiteres Beispiel einer syntaktischen Regel für einen übersichtlichen Kodierstil vor.

Fig. 3 zeigt schematisch die Schritte eines bevorzugten Ausführungsbeispiels des auf dem Computer 10 ausgeführten Verfahrens nach dem zweiten Aspekt der Erfindung. Zunächst wird wiederum das zu überprüfende Computerprogramm etwa von einer Diskette im Diskettenlaufwerk 14 in den RAM 16 geladen.

Entsprechend dem bereits erläuterten Verfahren liest die CPU 18 das zu untersuchende Computerprogramm aus dem Arbeitsspeicher 16 und durchsucht es auf Verletzungen von vorgegebenen Regeln der Programmiersprache und der weiteren Kodierregeln. Für jede gefundene Verletzung einer vorgegebenen Regel wird zunächst überprüft, ob diese Regelverletzung zu ignorieren ist.

Dies geschieht anhand einer Liste der zu ignorierenden Regelverletzungen. Diese Liste enthält die unterschiedlichen Definition für erlaubte Regelverletzungen, welche weiter unten genauer erläutert werden.

Ist die Rechtsverletzung zu ignorieren, so wird unmittelbar die nächste Regelverletzung gesucht.

Für jede gefundene und nicht zu ignorierende Verletzung einer vorgegebenen Regel wird mindestens eine mögliche Korrektur im Computerprogramm berechnet. Die Korrekturmöglichkeiten werden auf dem Monitor 20 oder über das Netzwerk zur interaktiven Auswahl durch einen Programmierer ausgeben.

Dabei kann der Programmierer wählen, ob er die Regelverletzung korrigieren oder ignorieren will.

Will er sie korrigieren, kann er aus den verschiedenen Korrekturmöglichkeiten mit Hilfe der Tastatur 12 oder einer (nicht gezeigten) Maus oder mittels Sprachsteuerung eine Korrekturmöglichkeit auswählen. Nach Wahl einer Korrekturmöglichkeit wird das zu überarbeitende Computerprogramm durch die Verarbeitungseinheit entsprechend geändert. Anschließend wird die nächste Regelverletzung gesucht.

Will der Programmierer die Regelverletzung ignorieren, so kann er zwischen einem einmaligen Ignorieren und einem ständigen Ignorieren wählen.

Wählt der Programmierer das einmalige Ignorieren, so wird die nächste Regelverletzung gesucht.

Wählt der Programmierer das ständige Ignorieren dieser Verletzung, so wird ihm eine Auswahl an möglichen Definitionen dieser Regelverletzung (siehe unten) angeboten. Hat er eine Definition gewählt, so wird diese in einer gesonderten Liste von Definitionen von zu ignorierenden Regelverletzungen gespeichert. Anschließend wird die nächste Regelverletzung gesucht.

Nach Abschluß der Korrekturen wird das Computerprogramm von der CPU 18 wieder in den Arbeitsspeicher 16 geschrieben, von wo aus es den diversen Ausgabemitteln zugeleitet werden kann.

Die Liste von Definitionen von zu ignorierenden Regelverletzungen kann zur Dokumentation ausgegeben werden.

Im bevorzugten Ausführungsbeispiel der Erfindung, der Korrektur von Regelverletzungen in VHDL (siehe unten), werden die zu ignorierenden Regelverletzungen auf unterschiedliche Weise alternativ oder kumulativ definiert.

1. Definieren von Ausnahmen auf Referenzbasis

Führt ein Name eines Objektes oder einer Funktion, der auf einen anderswo deklarierten Namen verweist, zu einer Regelverletzung, kann ein Ausnahmefall definiert werden durch

- eine konkrete Angabe des deklarierten Namens, oder
- die hierarchische Definition des Namens mittels der Bibliothek, zu der er gehört, der Designunit innerhalb der Bibliothek und schließlich des Namens innerhalb der Designunit.

Weiterhin kann eine Ausnahme durch Angabe einer Deklarationsumgebung definiert werden. Die Deklarationsumgebung kann zum einen eine Designunit sein. Sie kann auch eine Datei aus der Mehrzahl von Dateien sein, in die das Computerprogramm zerlegt wurde. Oder die Deklarationsumgebung kann ein beliebiger sichtbarer Deklarationsbereich sein, das ist die Zusammenschau aller durch explizites Einbinden als bekannt angegebener Bereiche aus anderen Bibliotheken oder Designunits.

Es kann die Definition einer Ausnahme auch durch Angabe lediglich eines Teils der oben erwähnten Bereiche erfolgen.

2. Definieren von Ausnahmen für Namen in lokaler Umgebung

Führt ein Name lokal zu einer Verletzung, so kann der Name konkret oder hierarchisch definiert werden. Der Name kann auch als zu einem bestimmten Bereich bzw. Umfeld eines Konstrukts gehörend spezifiziert werden.

3. Definieren von Ausnahmen in Gebieten des Quellcodes

Gebiete des Quellcodes können von der Überprüfung ausgenommen werden. Die Gebiete können definiert werden durch:

- Zeilen und/oder Spalten;

- Anfangszeilen und Endzeilen und/oder Anfangsspalten und Endspalten;
- Knoten im Parse-/Syntax-Baum, d.h. der Baumstruktur, die den grammatischen Aufbau des Quellcodes strukturiert widerspiegelt;
- Anfangs- und Endknoten im Parse-/Syntax-Baum;
- einem Pfad im Parse-/Syntax-Baum;
- Knoten mit Unterknoten im Parse-/Syntax-Baum; und/oder
- Knoten und/oder Unterknoten im Parse-/Syntax-Baum innerhalb eines ausgewählten Gebiets.

Auch können gewisse Klassen von Konstrukten von der Überprüfung bzw. Korrektur ausgenommen werden.

Im Folgenden werden anhand der Programmiersprache VHDL einige Beispiele für den Einsatz des Verfahrens aufgezeigt. VHDL steht für "very high speed integrated circuits hardware description language". Es ist eine objektbasierte Programmiersprache, die speziell zum Beschreiben und Testen von Hardwarebausteinen wie ASICs entwickelt wurde.

1. Zulassen unbenutzter Objekte

Bestimmte Simulationswerkzeuge erfordern es, daß zur Spezifikation von Attributen sogenannte Dummy-Objekte, d.h. nicht weiter benutzte Objekte, deklariert werden. Solche unbenutzten Objekte müssen im Quellcode verbleiben und dürfen nicht automatisch entfernt werden. Im folgenden, grob skizzierten Beispiel ist die Konstante c ein solches Dummy-Objekt.

```
P: process
  constant c : string := " ";
  attribute ...
begin
  -- ...
end f;
```

2. Namenskonvertierung

Oft wird externer Code in einen Quellcode eingebunden, der aus unterschiedlichen Gründen nicht vorgegebenen Kodierregeln entspricht, der aber auch nicht geändert werden soll. Es kann sich dabei z.B. um älteren Code oder zugekauften Code handeln. Werden definierte Konstrukte wie Typen, Objekte, etc. aus diesem Code verwendet, so zieht deren Benutzung automatisch die Meldung einer Verletzung einer Kodierregel nach sich, die bisher nicht verhindert werden konnte. In einem solchen Fall soll die Meldung der Verletzung unter Nennung der Kodierregel unterbleiben.

Als Beispiel diene die Kodierregel, gemäß derer Funktionen stets durch "_f" beendet werden und Konstanten durch "_c". Der nachfolgend beispielhaft abgedruckte fremde Quellcode verletzt diese Konvention:

```
function inc( number: Integer) return integer is
begin
  return number + 1;
end inc;
```

Die Einbindung dieser Funktion führt zu einer unumgänglichen Benutzung des nicht der Kodierregel entsprechenden Namens "inc" anstelle von "inc_f":

```
function add_f( number_c, slack_c : integer)
                                    return integer is
begin
  return inc( number_c + slack_c);
end add_f;
```

3. Zulassen von Konstrukten an bestimmten Stellen

Im Zusammenhang mit der Modellierung von Gated Clocks kann es notwendig sein, das Datum, welches an den Eingang eines getakteten Elements angelegt wird, etwas zu verzögern. Dies ist in der Regel durch Kodierregeln verboten. Im folgenden Beispiel wird der Eingang I um eine Femtosekunde (1 fs = 10^{-15} s) verzögert, indem der Eingang I auf die interne Variable T um 1 fs verzögert gegeben wird und anschließend T auf den Ausgang O gegeben wird.

```
Entity X1 is
  Port( Clock :in bit;
        Gate  :in bit;
        I     :in bit;
        O     :out bit);
End X1;

Architecture X2 of X1 is
  Signal Gated_clock : bit;
  Signal T: bit;
Begin
  Gated_clock <= Gate and Clock;
  T <= I after 1 fs when Clock = '1' and Clock'event;
  O <= T when Gated_clock = '1' and Gated_clock'event;
End X2;
```

Ein solches Konstrukt kann als Ausnahme erlaubt werden.

Im Rahmen der Erfindung sind zahlreiche Abwandlungen und Weiterbildungen sowohl der zu Fig. 2 als auch der zu Fig. 3 beschriebenen Beispiele möglich. Die vorgestellten Beispiele beschreiben nur einen Bruchteil der möglichen Ermittlungen von Korrekturen bzw. Ausnahmen von Korrekturen. Ferner können die beiden in den Fig. 2 und 3 beschriebenen Verfahrensabläufe kombiniert werden, wodurch erreicht wird, daß sowohl eine Ermittlung von Korrekturen als auch eine Ignorierung von be-

Patentansprüche

1. Verfahren zum Überarbeiten eines in einer Programmiersprache verfaßten Computerprogramms, wobei durch einen Computer
- das Computerprogramm zunächst auf Verletzungen von vorgegebenen Konsistenz-, Syntax-, Grammatik-Regeln oder lexikalischen Regeln durchsucht wird;
- für eine Verletzung einer vorgegebenen Regel eine mögliche Korrektur im Computerprogramm berechnet wird; und wobei
- das Computerprogramm gemäß der berechneten Korrektur geändert wird.

2. Verfahren nach Anspruch 1,
d a d u r c h g e k e n n z e i c h n e t, daß
für eine Verletzung einer vorgegebenen Regel eine Mehrzahl an möglichen Korrekturen der Regelverletzung im Computerprogramm berechnet wird.

3. Verfahren nach Anspruch 2,
d a d u r c h g e k e n n z e i c h n e t, daß
für eine Verletzung einer vorgegebenen Regel eine Korrekturmöglichkeit aus der Mehrzahl an Korrekturmöglichkeiten automatisch oder interaktiv ausgewählt wird.

4. Verfahren nach einem der Ansprüche 1 bis 3,
d a d u r c h g e k e n n z e i c h n e t, daß
das Computerprogramm bereits während einer sukzessiven Eingabe auf Verletzungen von vorgegebenen Regeln durchsucht wird und die Verletzungen noch während der Eingabe graphisch kenntlich gemacht werden.

5. Verfahren nach einem der Ansprüche 1 bis 4,
d a d u r c h g e k e n n z e i c h n e t, daß
das Computerprogramm bereits während einer sukzessiven Eingabe auf Verletzungen von vorgegebenen Regeln durchsucht wird

und eine vorgegebene Art von Verletzungen noch während der Eingabe automatisch korrigiert wird.

6. Computerprogrammprodukt, das direkt in den internen Speicher (16) eines Computers (10) geladen werden kann und Computerprogramm-Code-Abschnitte umfaßt, mit denen die Schritte nach einem der Ansprüche 1 bis 5 ausgeführt werden, wenn das Computerprogrammprodukt auf einem Computer ausgeführt wird.

7. Computerprogrammprodukt, das auf einem computergeeigneten Medium gespeichert ist und computerlesbare Programmittel umfaßt, die es einem Computer (10) ermöglichen, das Verfahren nach einem der Ansprüche 1 bis 5 auszuführen.

8. Datenträger, auf dem ein Computerprogramm gespeichert ist, das es einem Computer (10) ermöglicht, das Verfahren nach einem der Ansprüche 1 bis 5 auszuführen.

9. Computersystem mit Mitteln zum Ausführen des Verfahrens nach einem der Ansprüche 1 bis 5.

10. Computersystem zum Überarbeiten eines in einer Programmiersprache verfaßten Computerprogramms

- mit einer Speichereinrichtung (14, 16) zum Speichern des Computerprogramms auf einem Speichermedium;
- mit einer Verarbeitungseinheit (18), die das Computerprogramm aus der Speichereinrichtung (14, 16) ausließt und analysiert;
- wobei die Verarbeitungseinheit (18) das Computerprogramm zunächst auf Verletzungen von vorgegebenen Konsistenz-, Syntax-, Grammatik-Regeln oder lexikalischen Regeln durchsucht;
- wobei die Verarbeitungseinheit (18) für eine Verletzung einer vorgegebenen Regel eine mögliche Korrektur im Computerprogramm berechnet;
- wobei die Verarbeitungseinheit (18) das Computerprogramm gemäß der berechneten Korrektur ändert;

- wobei die Verarbeitungseinheit (18) anschließend das Computerprogramm in überarbeiteter Form in die Speichereinrichtung (14, 16) schreibt; und
- mit einer Ausgabeeinrichtung (20, 22), wobei die Ausgabeeinrichtung (20, 22) das Computerprogramm in überarbeiteter Form aus der Speichereinrichtung (14, 16) ausließt und ausgibt.

11. Computersystem nach Anspruch 10,
d a d u r c h g e k e n n z e i c h n e t, daß
die Verarbeitungseinheit (18) für eine Verletzung einer vorgegebenen Regel eine Mehrzahl an möglichen Korrekturen der Regelverletzung im Computerprogramm berechnet.

12. Computersystem nach Anspruch 11,
d a d u r c h g e k e n n z e i c h n e t, daß
die Verarbeitungseinheit (18) für eine Verletzung einer vorgegebenen Regel eine Korrekturmöglichkeit aus der Mehrzahl an Korrekturmöglichkeiten automatisch oder interaktiv auswählt.

13. Computersystem nach einem der Ansprüche 10 bis 12,
d a d u r c h g e k e n n z e i c h n e t, daß
die Verarbeitungseinheit (18) das Computerprogramm bereits während einer sukzessiven Eingabe auf Verletzungen von vorgegebenen Regeln durchsucht und die Verletzungen von der Ausgabeeinrichtung (18) noch während der Eingabe graphisch kenntlich gemacht werden.

14. Computersystem nach einem der Ansprüche 10 bis 13,
d a d u r c h g e k e n n z e i c h n e t, daß
die Verarbeitungseinheit (18) das Computerprogramm bereits während einer sukzessiven Eingabe auf Verletzungen von vorgegebenen Regeln durchsucht und eine vorgegebene Art von Verletzungen noch während der Eingabe automatisch korrigiert.

15. Verfahren zum Überarbeiten eines in einer Programmiersprache verfaßten Computerprogramms, wobei durch einen Computer

- das Computerprogramm auf Verletzungen von vorgegebenen Konsistenz-, Syntax-, Grammatik-Regeln oder lexikalischen Regeln hin analysiert wird; und wobei
- Verletzungen definiert werden können, die bei der Analyse automatisch ignoriert werden.

16. Verfahren nach Anspruch 15,
d a d u r c h g e k e n n z e i c h n e t, daß
die Definition von zu ignorierenden Verletzungen durch konkrete, verallgemeinerte oder hierarchische Spezifizierung der Verletzung erfolgt.

17. Verfahren nach Anspruch 15 oder 16,
d a d u r c h g e k e n n z e i c h n e t, daß
die Definition von zu ignorierenden Verletzungen durch Angabe der Deklarationsumgebung der Verletzung erfolgt.

18. Verfahren nach einem der Ansprüche 15 bis 17,
d a d u r c h g e k e n n z e i c h n e t, daß
die Definition von zu ignorierenden Verletzungen durch Spezifizierung eines Bereichs oder Umfelds eines Konstrukts erfolgt.

19. Verfahren nach einem der Ansprüche 15 bis 18,
d a d u r c h g e k e n n z e i c h n e t, daß
die Definition von zu ignorierenden Verletzungen durch Spezifizieren von Gebieten des Quellcodes des Computerprogramms erfolgt, wobei die Gebiete definiert werden durch die Angabe von:

- Zeilen und/oder Spalten;
- Anfangszeilen und Endzeilen und/oder Anfangsspalten und Endspalten;
- Knoten im Parse-/Syntax-Baum;
- Anfangs- und Endknoten im Parse-/Syntax-Baum;

- einem Pfad im Parse-/Syntax-Baum.

20. Verfahren nach einem der Ansprüche 15 bis 19, dadurch gekennzeichnet, daß die Definition von zu ignorierenden Verletzungen durch Angabe einer Klasse von Konstrukten erfolgt.

21. Verfahren nach einem der Ansprüche 15 bis 20, dadurch gekennzeichnet, daß die Definition von zu ignorierenden Verletzungen durch Angabe einer Klasse von Knoten erfolgt.

22. Verfahren nach Anspruch 21, dadurch gekennzeichnet, daß die Definition von zu ignorierenden Verletzungen durch Angabe einer Klasse von Knoten mit Unterknoten erfolgt.

23. Computerprogrammprodukt, das direkt in den internen Speicher (16) eines Computers (10) geladen werden kann und Computerprogramm-Code-Abschnitte umfaßt, mit denen die Schritte nach einem der Ansprüche 15 bis 22 ausgeführt werden, wenn das Computerprogrammprodukt auf einem Computer ausgeführt wird.

24. Computerprogrammprodukt, das auf einem computergeeigneten Medium gespeichert ist und computerlesbare Programmittel umfaßt, die es einem Computer (10) ermöglichen, das Verfahren nach einem der Ansprüche 15 bis 22 auszuführen.

25. Datenträger, auf dem ein Computerprogramm gespeichert ist, das es einem Computer (10) ermöglicht, das Verfahren nach einem der Ansprüche 15 bis 22 auszuführen.

26. Computersystem mit Mitteln zum Ausführen des Verfahrens nach einem der Ansprüche 15 bis 22.

27. Computersystem zum Überarbeiten eines in einer Programmiersprache verfaßten Computerprogramms
- mit einer Speichereinrichtung (14, 16) zum Speichern des Computerprogramms auf einem Speichermedium;
- mit einer Verarbeitungseinheit (18), die das Computerprogramm aus der Speichereinrichtung (14, 16) ausliest und analysiert;
- wobei die Verarbeitungseinheit (18) das Computerprogramm zunächst auf Verletzungen von vorgegebenen Konsistenz-, Syntax-, Grammatik-Regeln oder lexikalischen Regeln durchsucht; und
- mit Mitteln zum Definieren von Verletzungen, die bei der Analyse automatisch ignoriert werden.

28. Computersystem nach Anspruch 27,
g e k e n n z e i c h n e t d u r c h
Mittel zum Definieren von zu ignorierenden Verletzungen durch konkrete, verallgemeinerte oder hierarchische Spezifizierung der Verletzung.

29. Computersystem nach Anspruch 27 oder 28,
g e k e n n z e i c h n e t d u r c h
Mittel zum Definieren von zu ignorierenden Verletzungen durch Angabe der Deklarationsumgebung der Verletzung.

30. Computersystem nach einem der Ansprüche 27 bis 29,
g e k e n n z e i c h n e t d u r c h
Mittel zum Definieren von zu ignorierenden Verletzungen durch Spezifizierung eines Bereichs oder Umfelds eines Konstrukts.

31. Computersystem nach einem der Ansprüche 27 bis 30,
g e k e n n z e i c h n e t d u r c h
Mittel zum Definieren von zu ignorierenden Verletzungen durch Spezifizieren von Gebieten des Quellcodes des Computerprogramms, wobei die Gebiete definiert werden durch die Angabe von:
- Zeilen und/oder Spalten;

- Anfangszeilen und Endzeilen und/oder Anfangsspalten und Endspalten;
- Knoten im Parse-/Syntax-Baum;
- Anfangs- und Endknoten im Parse-/Syntax-Baum;
- einem Pfad im Parse-/Syntax-Baum.

32. Computersystem nach einem der Ansprüche 27 bis 31, gekennzeichnet durch Mittel zum Definieren von zu ignorierenden Verletzungen durch Angabe einer Klasse von Konstrukten.

33. Computersystem nach einem der Ansprüche 27 bis 32, gekennzeichnet durch Mittel zum Definieren von zu ignorierenden Verletzungen durch Angabe einer Klasse von Knoten.

34. Computersystem nach Anspruch 33, gekennzeichnet durch Mittel zum Definieren von zu ignorierenden Verletzungen durch Angabe einer Klasse von Knoten mit Unterknoten.

Zusammenfassung

Verfahren zum Überarbeiten eines in einer Programmiersprache verfaßten Computerprogramms

Bei dem erfindungsgemäßen Verfahren wird der Quellcode eines Computerprogramms zunächst auf Verletzungen von vorgegebenen Konsistenz-, Syntax- oder Grammatik-Regeln oder lexikalischen Regeln durchsucht. Für eine Verletzung einer vorgegebenen Regel wird eine mögliche Korrektur berechnet. Anschließend wird der Quellcode des Computerprogramms gemäß der berechneten Korrektur automatisch oder interaktiv geändert. Alternativ können Verletzungen definiert werden, die bei der Analyse automatisch ignoriert werden.

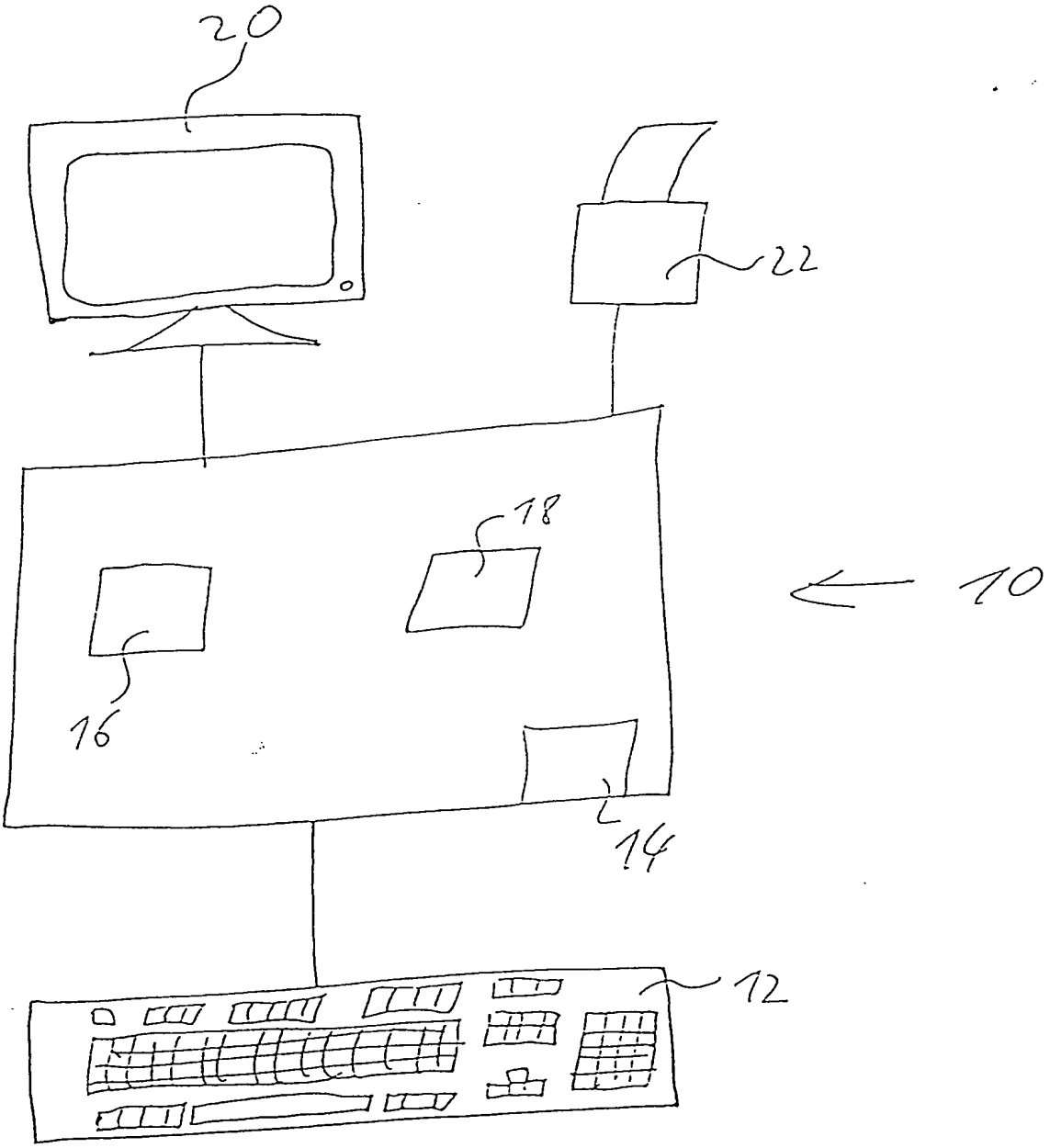


Fig. 1

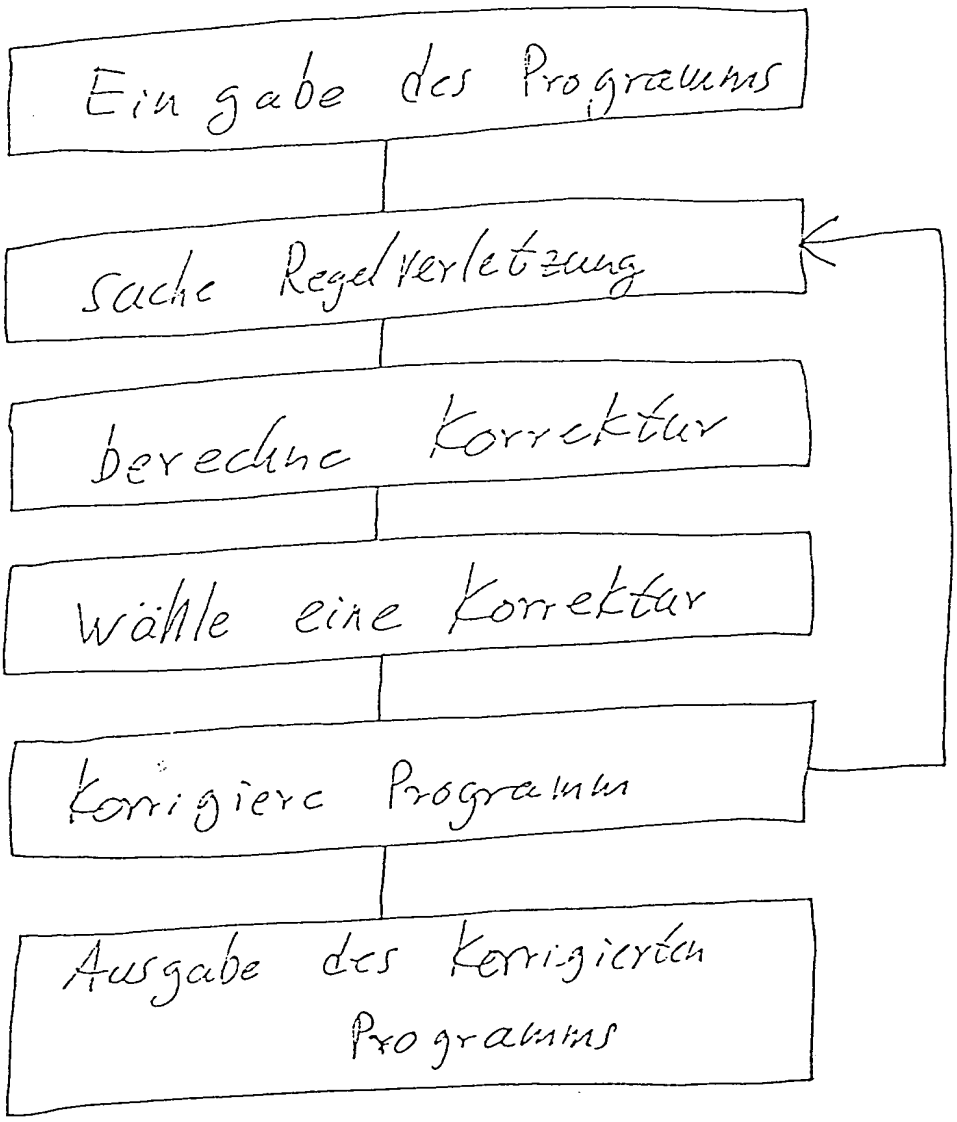

Eingabe des Programms
suche Regelverletzung
berechne Korrektur
wähle eine Korrektur
korrigiere Programm
Ausgabe des korrigierten Programms

Fig. 2

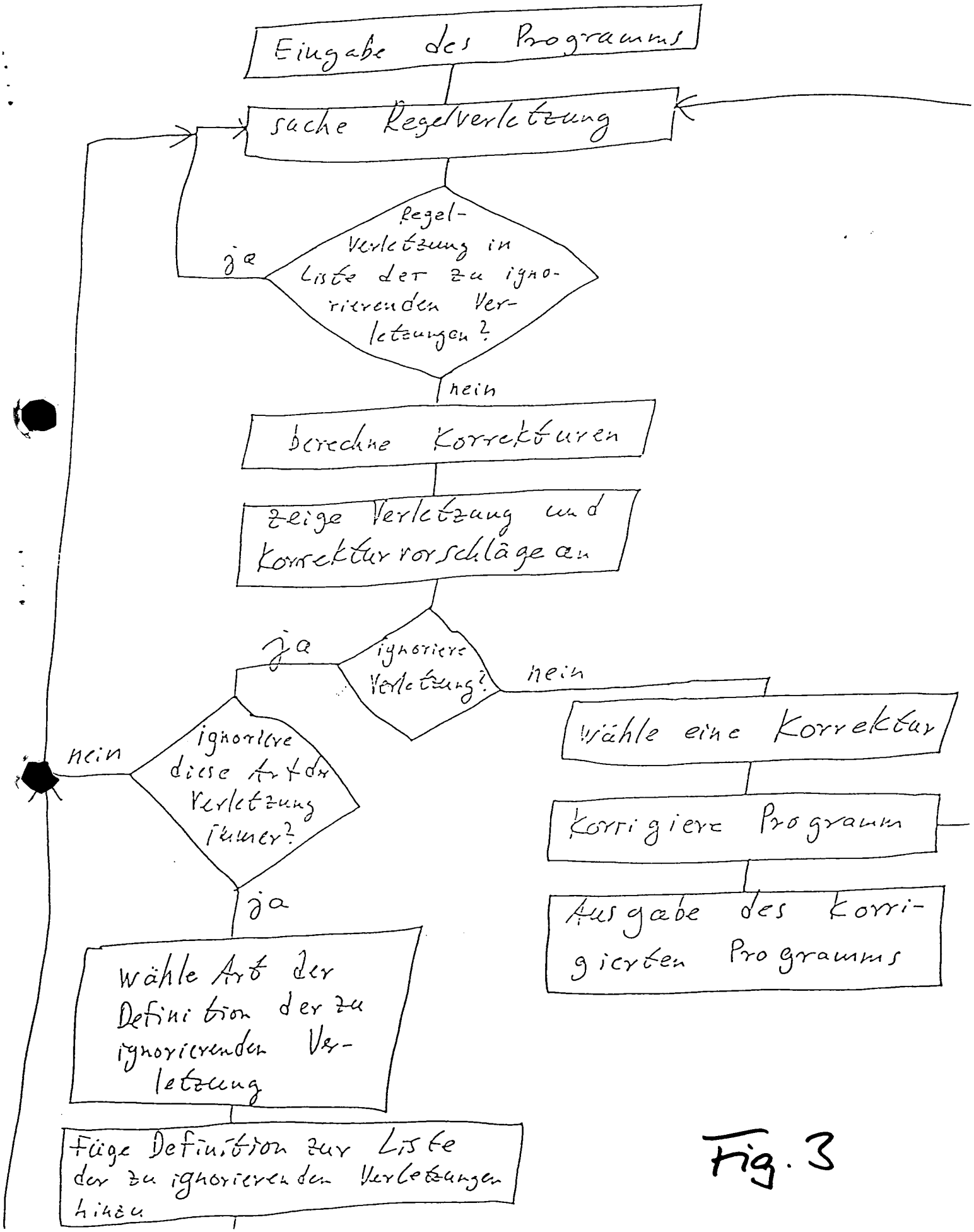
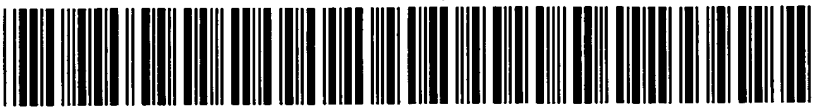

Eingabe des Programms
suche Regelverletzung
Regel-Verletzung in Liste der zu ignorierenden Verletzungen?
ja
nein
berechne Korrekturen
zeige Verletzung und Korrekturvorschläge an
ignoriere Verletzung?
ja
nein
ignoriere diese Art der Verletzung immer?
nein
ja
wähle eine Korrektur
korrigiere Programm
Ausgabe des korrigierten Programms
Wähle Art der Definition der zu ignorierenden Verletzung
Füge Definition zur Liste der zu ignorierenden Verletzungen hinzu

Fig. 3

US 0993535504P1



Creation date: 10-15-2004
Indexing Officer: AGOMEZ - ALFREDO GOMEZ, JR.
Team: OIPEBackFileIndexing
Dossier: 09935355

Legal Date: 09-26-2001

No.	Doccode	Number of pages
1	IDS	2
2	PA..	1

Total number of pages: 3

Remarks:

Order of re-scan issued on .....